CoLink: A Programming Framework for Decentralized Data Science
Project Proposal (Group Name: CoLink Dev)

Xiaoyuan Liu
xiaoyuanliu@berkeley.edu

Tianchen Liu
tianchenliu@berkeley.edu

Bryce Wong
brywong@berkeley.edu

1 Introduction

Data collaboration is a common need in many research fields including finance, health care, cybersecurity, and many others. To protect data privacy in such collaborations, many existing protocols and systems support certain types of computations without sharing the original data input across different participants. However, there are many deployment challenges for using existing solutions in the real world and there lacks a universal framework with a consistent data collaboration workflow.

In this work, we propose to investigate an efficient programming abstraction to support decentralized data science. It provides a unified interface to manage user, storage, communication, and computation. Extending gRPC, we propose to simplify the development of multiparty protocols and orchestrate implementations in different programming languages to work together consistently. With a unified interface that increases potential data contributors, our framework has the potential to enable larger-scale decentralized data collaboration.

CoLink targets to simplify the deployment of decentralized data science solutions. Two types of programmers can benefit from CoLink. Data collaboration *application developers* who understand the functionalities and threat models of security protocols can directly integrate off-the-shelf protocols as black boxes into their application system. And *protocol designers* who are often researchers or cryptography experts save efforts on building repeated basic abstractions and writing reusable protocol implementation code.

2 Design

We propose to build CoLink to accelerate the development of security protocols and overcome the deployment obstacles. In this section, we identify three major types of deployment difficulties, specify design principles, and discuss CoLink framework architecture.

2.1 Deployment Difficulties

- **Practicability.** Compared with protocol simulations that focus on correctness and performance evaluation, a real-world privacy-preserving system needs great engineering efforts to be practical. In most solutions, different participants need to be able to discover and identify each other. They often need to build secure and authenticated connections for communication, store and transmit data, and perform various computations. Building such a system requires the management of authenticated users and their roles, database, network communication, computation runtime, and many other resources.

- **Compatibility.** The implementation of one protocol often requires the functionality of many other protocols. For example, federated learning often requires secure aggregation as its building block to merge the gradients in a privacy-preserving way. Secure aggregation might further require synchronization for randomness. However, different protocols are often implemented with different library dependencies in different programming languages, making it hard for integration. Even protocols implemented in the same programming language might use different data communication formats. Poor compatibility not only slows down the integration but also makes the replacement and upgrade of the protocol with similar functionality difficult.

- **Publicity.** Knowing and understanding the terms to describe the protocol requires comprehensive background knowledge in the field of cryptography. Sometimes there are multiple implementations available with various differences that can be only discovered after testing. There lacks a unified way to describe the functionalities and security guarantees of various protocols and an index that supports searching and comparison for them.

2.2 CoLink Design

Observing these difficulties, we plan to practice the following principles in our framework design.

- **Minimalism.** The programming abstraction should be easy to learn and use. We should reduce the number of concepts and interfaces one needs to learn and make the abstraction flexible enough to solve the practicability issue. We should also make our implementation lightweight reducing resource consumption and making security audits easier. For functionalities that already exists in other systems, we should directly reuse mature plan instead of implementing our own version.
- **Extensible.** Because multi-party data collaboration is a generic abstraction for various tasks, we should separate a small set of core functionalities and reuse the collaboration computation abstraction to support additional features. Treating additional functionalities as extensions also makes the framework highly configurable. By managing extensions with an index, we increase publicity of various privacy-preserving protocols.
- **Reusable.** Framework should allow comprehensive interactions among different extensions. Ideally, developers should be able to call certain protocols without knowing implementation details. When implementing new protocols, one should be able to reuse existing protocols as building blocks.

Following the design principles, we design a plug-in architecture for CoLink shown in Figure 1. At a high level, each participant client has a *CoLink server* hosted on their laptop or in a trusted cloud environment and store their data there. CoLink servers manage the storage and data collaboration requests, ask the client to make decisions on whether to approve a collaboration request, and communicate with other CoLink servers to run protocols on behalf of their owner.

Each user's framework contains three components:

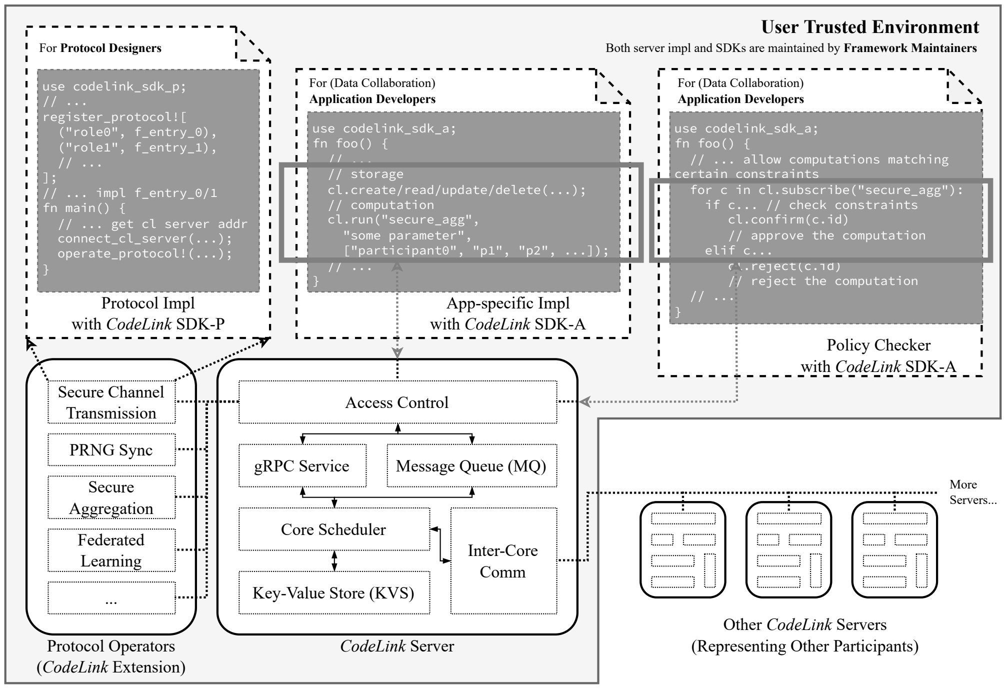
- **CoLink Server.** To use CoLink in the programming environment, each client needs to setup, in an environment they trust, a CoLink server that manages the user, storage, and network-related abstractions. For scalability, multiple clients may share the same CoLink server as the server provides user-based isolation. Inside the CoLink server, there is a key-value store for persistent storage, a core scheduler to handle various requests, an inter-core communication module to connect with other CoLink servers, a gRPC service, and a message queue to handle connections from both the client and other CoLink servers with an additional layer of access control.
- **CoLink SDKs.** To simplify the programming experience for both application developers and protocol designers as mentioned in Section 1, we provide SDKs in various programming languages to help developers write programs that can interact with the CoLink Server. Specifically, for each language, we target to provide two different SDKs: *an application SDK (SDK-A)* for application developers and *a protocol SDK (SDK-P)* for protocol designers. SDK-A allows the client to update storage, manage computation requests, and monitor CoLink server status. And SDK-P allows protocol designers to write CoLink Extensions that extend the functionality of CoLink to support new protocols.
- **Protocol Operators (CoLink Extensions).** According to our design principle of being extensible, we detach the implementation of specific protocol from the CoLink server. With SDK-P, a protocol designer can write a program that connects to the CoLink server to register itself as a protocol operator. Whenever someone requests a protocol execution from the CoLink server and gets approved, the CoLink server forwards the request to the corresponding operator.

With the above three components, CoLink provides the necessary programming abstraction to handle computation and can link code in different programming languages from different participants together. By providing SDK-P in different programming languages, CoLink can integrate protocols implemented in different languages as operators. By having SDK-A in different programming languages, the client can use whatever environment they prefer to start certain protocol executions. We plan to finalize the workflow to orchestrate a multi-party data collaboration in our next submission.

3 Implementation

In addition to a paper finalizing our design and the presentation, we also target to produce:

1. A prototype CoLink server implemented in Rust. We will select dependencies for this prototype and use them to demonstrate the feasibility of our design. We also target to open-source our solution. We welcome other teams who target protocol design-related research to try out CoLink near the end of the course.
2. An SDK-P in Rust and an SDK-A in both Rust and Python. We will use it to demonstrate the cross-language protocol execution feature.


User Trusted Environment
Both server impl and SDKs are maintained by Framework Maintainers
For Protocol Designers
Protocol Impl with CodeLink SDK-P
For (Data Collaboration) Application Developers
App-specific Impl with CodeLink SDK-A
For (Data Collaboration) Application Developers
Policy Checker with CodeLink SDK-A
Secure Channel Transmission
PRNG Sync
Secure Aggregation
Federated Learning
...
Protocol Operators (CodeLink Extension)
Access Control
gRPC Service
Message Queue (MQ)
Core Scheduler
Inter-Core Comm
Key-Value Store (KVS)
CodeLink Server
More Servers...
Other CodeLink Servers (Representing Other Participants)

```
use codelink_sdk_p;
// ...
register_protocol![
  ("role0", f_entry_0),
  ("role1", f_entry_1),
  // ...
];
// ... impl f_entry_0/1
fn main() {
  // ... get cl server addr
  connect_cl_server(...);
  operate_protocol!(...);
}
```

```
use codelink_sdk_a;
fn foo() {
  // ...
  // storage
  cl.create/read/update/delete(...);
  // computation
  cl.run("secure_agg",
    "some parameter",
    ["participant0", "p1", "p2", ...]);
  // ...
}
```

```
use codelink_sdk_a;
fn foo() {
  // ... allow computations matching
certain constraints
  for c in cl.subscribe("secure_agg"):
    if c... // check constraints
       cl.confirm(c.id)
       // approve the computation
    elif c...
       cl.reject(c.id)
       // reject the computation
  // ...
}
```

Figure 1: CoLink Architecture.

3. A case study with a basic secure aggregation protocol. We will compare the programming difficulty of writing a basic secure aggregation application with or without CoLink.
4. A project homepage providing tutorial, documentation, and a list of supported protocols for publicity.

4 Evaluation

1. To evaluate the performance of CoLink, we will conduct log-based trace analysis for performance profiling and analyze the performance overhead introduced by CoLink for certain use cases.

2. We will conduct a usability case study for both secure aggregation example mentioned in Section 3 and protocols from other course groups.

5 Discussion

Based on the team's previous experience in building privacy-preserving systems, we think the current system design will be helpful to both application developers and cryptography protocol designers. Although we do not target to integrate many protocols in the current development stage. For future work, however, we are not certain how hard it would be to integrate existing building blocks (e.g. the ones mentioned in [1, 2]) into our framework. We will continue to evaluate the programming difficulty of integrating various existing efforts during our project development.

To further increase the publicity mentioned in Section 1, in the future we also plan to write CoLink protocol that acts as a data registry. Such a registry can be helpful in the client discovery procedure and further simplify the programming abstraction. We will also explore the possibility of using web3 technology for the registry.

6 Project Milestones and Planned Timeline

We specify the three key stages of the project as follows.

1. **Mar 24.** Finalize system design. Start implementing the CoLink server.
2. **Apr 14.** Finish an early prototype CoLink server, an example SDK, and the basic secure aggregation.
3. **May 5.** Prepare the prototype CoLink. Finish the case study and performance evaluation. After this stage, we will focus on writing the final report.

7 Division of labor

Xiaoyuan will work on the system design, implement specific protocols, and help with the case study and building the homepage. Tianchen will work on the system implementation, protocol packing, and help with the protocol implementation. Bryce will work on the interaction workflow design, system UI, and help with the system and protocol implementation.

References

[1] Awesome homomorphic encryption. `https://github.com/jonaschn/awesome-he`. 2022-03-10.

[2] Awesome MPC. `https://github.com/rdragos/awesome-mpc`. 2022-03-10.